1

Introduction to iPhone Development with MonoTouch for C# Developers

WHAT'S IN THIS CHAPTER?

- The history of the iPhone and its mindshare
- A short history of Mono and its relationship to the .NET Framework
- How MonoTouch opens the iPhone to .NET Developers
- Why MonoTouch is so attractive to developers

The past several years have seen an amazing growth in the use of smartphones, and *USA Today* recently reported how smartphones have become an indispensable part of people's lives.

Although Windows-based computers running 32-bit x86 or 64-bit x64 processors dominate the desktop computer marketplace, and the .NET Framework is the dominant development environment for the Windows platform, no single vendor or platform dominates the mobile device marketplace; devices based on Symbian, Research in Motion (Blackberry), Windows Mobile, Android, and other platforms are available. In addition, devices may run the same operating system and be presented to the user in separate form factors. This fracture in the marketplace is problematic for developers — how can they take a development framework, or tool, that they already know and use that knowledge in a device that has a large and growing market share?

This chapter looks at how the largest segment of developers can target the smartphone with the highest mindshare, and that the smartphone is growing faster in marketshare than any other device.

PRODUCT COMPARISON

This section takes a quick look at .NET Framework, Mono and MonoTouch — three products that have allowed the largest segment of developers to target the iPhone, the most exciting mobile platform currently in the marketplace.

.NET Framework

In the late 1990s, Microsoft began work on the .NET Framework. The first version of the framework shipped in 2002. Microsoft proceeded to introduce subsequent versions of the .NET Framework and has recently introduced the .NET Framework 4. The .NET Framework comes in various versions, including 32-bit versions, 64-bit versions, a version for the XBOX gaming platform, and a version for Microsoft's mobile devices referred to as the Compact Framework (CF). A few facts about .NET Framework:

- Microsoft released a development tool, *Visual Studio .NET*, with the Framework. This tool is the Integrated Development Environment for .NET.
- It's based on a virtual machine that executes software written for the framework. This virtual machine environment is referred to as the *Common Language Runtime (CLR)*, and it is responsible for security, memory management, program execution, and exception handling.
- Applications written in the .NET Framework are initially compiled from source code, such as Visual Basic or C#, to an intermediate language, called MSIL. The initial compilation is performed by calling the language specific command line compiler, Visual Studio, or some other build tool. A second compilation set is typically done when an application is executed. This second compilation takes the intermediate language and compiles it into executable code that can be run on the operating system. This second compilation is referred to as *just-in-time compilation*.
- It's language independent, and numerous languages are available for the Framework. In the Visual Studio, Microsoft has shipped various languages including Visual Basic, F#, C++, and C#.
- It has a series of libraries that provide consistent functionality across the various languages. These libraries are referred to as the *Base Class Libraries*.
- Microsoft has submitted various parts of the .NET Framework to various standard organizations. Some of these are the C# language, the Common Language Infrastructure, Common Type System (CTS), Common Language Specification (CLS), and Virtual Execution System (VES).
- It has the largest number of developers for any development framework out there. As a result, more developers are familiar with the .NET Framework than any other development framework.
- A disadvantage of the .NET Framework is that it is not available for non-Microsoft platforms.

Mono

Mono is an open source project that provides a C# compiler and Common Language Runtime on non-Windows operating systems. Mono is currently licensed under GPL version 2, LGPL version 2, the MIT, and dual licenses. Mono runs on Mac, Linux, BSD, and other operating systems.

Mono was officially announced in 2001 and is the brainchild of Miguel de Icaza. Mono version 1.0 shipped in 2004, and currently Mono is at Version 2.6. Mono continues to be led by Miguel de Icaza and is under the general leadership and support of Novell.

As much as there is the desire to match the .NET Framework's features, this is not possible due to the fact that Microsoft has more resources and a head start in the development of those features. At the same time, the Mono project has parity with a large number of .NET Framework features.

Along with Mono, there is an open source IDE called *MonoDevelop*, which started as a port of the SharpDevelop IDE. MonoDevelop began as a project to allow for Mono development on Linux, but with the release of MonoDevelop 2.2, the ability to develop with Mono expanded to the Mac, Windows, and several other non-Linux UNIX platforms.

Though the .NET Framework is very popular, two issues make it unsuitable for running on the iPhone:

- At some level Apple and Microsoft are competitors and are likely not too excited to work together.
- The .NET Framework fundamentally is dynamically compiled at runtime. This is the just-in-time compilation of the .NET Framework. This is a violation of the Apple license and the operating principles of the iPhone OS.

Given that code running on the Microsoft .NET Framework is compiled to machine code at runtime using the just-in-time compilation, one would expect that applications written for Mono would have the same behavior and thus not be suitable for running on the iPhone. However, Mono has a technology that allows for applications to be compiled ahead of time, referred to as *AOT technology*.

A disadvantage of .NET/Mono and the iPhone is that .NET/Mono developers cannot take their .NET/Mono/C# knowledge and apply it to the iPhone platform. As illustrated in Figure 1-1, you see that the reason .NET/Mono developers can't target the iPhone is because they're two separate entities.

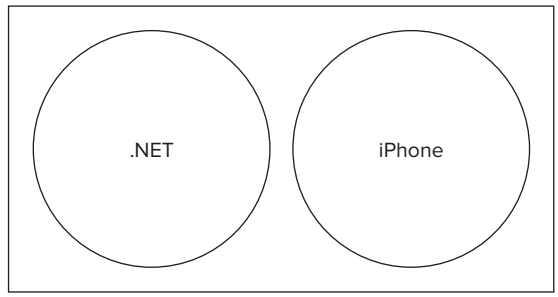

FIGURE 1-1

MonoTouch

In 2009, Novell announced and shipped MonoTouch, which allows .NET developers to create native iPhone applications in C#. With MonoTouch, applications are compiled into executable code that runs on the iPhone. The significance of this should not be understated: .NET/Mono developers can target the iPhone through MonoTouch. This is illustrated in Figure 1-2.

How does MonoTouch accomplish this? Does it somehow allow Windows Forms applications to be translated or recompiled and deployed on the iPhone? MonoTouch provides a

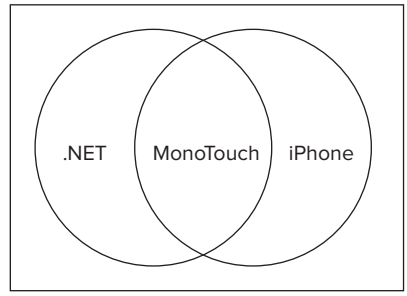

FIGURE 1-2

.NET layer over the native iPhone programming layer present on the iPhone OS, referred to as *Cocoa Touch*. Cocoa Touch is based on the Cocoa layer in the Mac OS X and is available on the iPhone, iPod Touch, and the iPad. MonoTouch does not provide a mechanism to cross-compile Windows Forms applications, but allows developers to build applications that run natively on the iPhone.

Overall, the application programming interface (API) exposed by the MonoTouch SDK is a combination of the .NET 2.0 Framework's core features, the Silverlight 2.0 API, and the APIs on the iPhone. MonoTouch provides a bridge (interop) between the iPhone's native APIs based on Objective-C and C-based APIs to the .NET world that C# developers are accustomed to.

MonoTouch Components

MonoTouch is made up of the following four components:

- The `Monotouch.dll` is a C# assembly that provides a binding API into the iPhone's native APIs.
- A command-line tool that compiles C# and Common Intermediate Language (CIL) code. This compiled code can then be run in the simulator or an actual iPhone.
- An add-in to MonoDevelop that allows for iPhone development and for Interface Builder to create graphical applications.
- A commercial license of the Mono runtime, which allows for the static linking of the Mono runtime with the code developed.

Namespaces and Classes

MonoTouch provides a rich set of namespaces and classes to support building applications for the iPhone. Some of the most popular namespaces and classes are:

- **MonoTouch.ObjCRuntime:** This namespace provides the interop/bridge between the .NET/C# world and the Objective-C world of the iPhone.
- **MonoTouch.Foundation:** This namespace provides support for the data types necessary to communicate with the Objective-C world of the iPhone. Most types are directly mapped. For example, the `NSObject` Objective-C base class is mapped to the `MonoTouch.Foundation.NSObject` class in C#. Some classes are not directly mapped and are instead mapped to their native .NET types. For example, `NSString` maps to the basic string type and `NSArray` maps to a strongly typed array.
- **MonoTouch.UIKit:** This namespace provides a direct mapping between the UI components within Cocoa Touch. The mapping is done by providing .NET classes for each UI component, and this is the namespace that developers will likely spend most of their time working with. For .NET developers, Cocoa Touch is an abstraction layer or API for building programs that run in the iPhone. Cocoa Touch is based on the Cocoa API used in building programs that run on the Mac OS X operating system. Cocoa Touch can be thought of as Cocoa tuned for the touch-based iPhone operating system.

- **OpenTK:** This namespace is a modified version of the OpenTK API. OpenTK is an object-oriented binding for OpenGL, which stands for the Open Graphics Library. OpenGL is an API for using three-dimensional graphics. OpenTK is a library for performing OpenGL, OpenAL, and OpenCL. It is written in C# and runs on Windows, Mac OS X, and Linux. The OpenTK implementation on the iPhone has been updated to use CoreGraphics and to only expose the functionality available on the iPhone.

In addition, MonoTouch provides a set of additional namespaces that may be important to you. These are:

- `MonoTouch.AddressBook`
- `MonoTouch.CoreGraphics`
- `MonoTouch.AddressBookUI`
- `MonoTouch.AudioToolbox`
- `MonoTouch.MapKit`
- `MonoTouch.MediaPlayer`
- `MonoTouch.AVFoundation`
- `MonoTouch.MediaPlayer`
- `MonoTouch.CoreAnimation`
- `MonoTouch.SystemConfiguration`

These namespaces are fairly self-explanatory in their functionalities and are specific to the iPhone.

MonoDevelop

MonoDevelop is a free IDE used for developing with Mono and is an early branch of the SharpDevelop IDE. Originally, MonoDevelop ran only on Linux, but with version 2.2, MonoDevelop began running on the Mac. MonoDevelop on the Mac allows for the creation and management of iPhone projects as well as debugging and deployment to the simulator and devices for testing.

iPhone

There's no doubt that Apple has changed the mobile device marketplace since the introduction of the original iPod in 2001. Although the iPod was not the first device to play mp3 files, it was the first product that played mp3 files, made it easy to use, and provided an easy-to-use marketplace to purchase audio files. The iPod really caused the mp3 device marketplace to explode.

In January 2007, Apple turned the smartphone upside down when it officially announced the first-generation iPhone. The iPhone was designed to be a smartphone that provided web browsing,

e-mail, and multimedia capabilities. The first-generation iPhone connected to a wireless network and applications were delivered to the user over the mobile version of Safari.

Writing a web-based application for the iPhone is fairly simple. The Safari web browser is a great tool — it does an excellent job of scaling web-based applications to run an iPhone-sized screen. It also does well running applications that are highly dependent on JavaScript. Upgrading an iPhone web-based application is also a simple matter of deploying a new version of the application to a web server. Many applications have taken this approach.

Unfortunately, web applications are not suitable for all applications — applications that require some background processing, access to local resources, must work when a network connection is unavailable, and some other application types don't work well in this model.

So, the question becomes how does one write an application that fits into the iPhone?

The first-generation iPhone did not have support for users to load applications on the device. For a few users, this was not acceptable, and they began *jailbreaking* their iPhones, which is the process where users run software on their devices that Apple has not approved.

Jailbreaking has several problems:

- **Technical Issues:** Jailbreaking requires the iPhone's owner to perform the operation, and many iPhone users are not technically proficient enough to do this.
- **Legality:** The legality of jailbreaking is unclear at the time of this writing. It is not clear where jailbreaking falls within the Digital Millennium Copyright Act. The Electronic Frontier Foundation has asked the United States Copyright Office to recognize an exception to the DMCA that allows iPhone owners to jailbreak their devices. Apple has argued in response that jailbreaking an iPhone is a copyright violation.
- **Unknowns:** It comes with a series of unknowns. How well can a jailbroken iPhone be upgraded to new versions of the iPhone operating system (OS)? Will jailbreaking an iPhone open it up to security issues?

In 2008, Apple introduced the second generation of the iPhone, referred to as the iPhone 3G. With this generation and the new version of the iPhone OS, Apple released a number of enhancements, including the ability to run applications natively on the device. In addition to this, Apple has put together an ecosystem whereby users can find and install applications on their iPhone device called the App Store.

These native applications are a great improvement over web-based applications, which are limited in what they can do on a device. Fundamentally, they have to be loaded over the Web and are not able to access all device features. Native applications tend to have more support for device features like the accelerometer, file system, camera, cross-domain web services, and other features that are outside of features available in HTML and JavaScript. In addition, native applications do not depend on the wireless network to be loaded, whereas a web application is dependent on the wireless network for loading.

In 2009, Apple introduced the iPhone 3GS and version 3 of the iPhone operating system. The iPhone 3GS, a refinement of the iPhone 3G, supports higher data rates than the iPhone 3G, an improved camera, an updated CPU, and voice control.

In 2010, Apple announced and shipped the iPad. The iPad is a tablet device, and it has a larger screen than the iPhone. Also significant is that it shipped with the iPhone operating system that is fundamentally different than the iPhone.

Along with the release of each new iPhone, Apple has introduced a new iPod touch. The iPod touch can be thought of as an iPhone without the phone, camera, and support for the 3G data services; however, the iPod touch does have support for wireless networking using WiFi.

Since its availability three years ago, Apple has shipped more than 60 million units of the iPhone. The iPad is estimated to ship several million units of the iPad in its first year of availability, and this will likely result in the iPad being the most popular tablet in 2010.

Unfortunately, for developers, three issues must be considered when running on the device:

- The iPhone operating system does not allow for software code that is interpreted or dynamically compiled in any way.
- Apple's licensing for the SDK and developing with the iPhone does not allow for applications to have interpreted or dynamically compiled code.
- Apple has an extensive validation process for iPhone applications. Some of the automated tests for an application will check for dynamically compiled and interpreted code.

These issues and licensing are something that developers need to be knowledgeable of, and somewhat limit the choices that a developer has for writing applications that run on the iPhone.

MOBILE DEVELOPMENT

There are a few things developers need to know when building applications on the iPhone with MonoTouch:

- The iPhone has a startup timer. If an application takes longer than 20 seconds to start up, the iPhone OS kills it.
- The iPhone OS will kill any application that is unresponsive for longer than 20 seconds while the application is running. To work around this, you need to perform some type of asynchronous operation.
- The time spent processing the `FinishedLaunching()` event counts against the startup timer. As a result, you do not want any long-term synchronous processing in the `FinishedLaunching()` event.
- The iPhone simulator is good for initial testing; however, it is not necessarily accurate for all testing. Just because something works in the simulator doesn't mean it will run in the iPhone in the same way. Final testing should be completed in the iPhone.

- With .NET, executables are fairly small. Every application shares the .NET Framework, so the applications don't have their own copy of the framework. MonoTouch is not built into the iPhone and its applications must have their own copy of the framework; MonoTouch is compiled into your application. The result is that MonoTouch applications are larger on disk than a comparable Objective-C application.

Although MonoTouch is a commercially licensed product, it is still a product that is under continual development, and MonoTouch may not have support for a specific namespace or assembly. You have two options for this situation:

- Wait on the implementation of that assembly from the MonoTouch product.
- Pull the necessary code or assembly into to your project. This is fairly common if the application needs to use code within the `System.Web.*` namespaces.

In addition to the technical issues of building an application for the iPhone, some design issues that developers should be aware of include:

- Don't design an application for a desktop environment and think that it can be scaled down to an iPhone, or any mobile device. An iPhone does not have the display, hardware, or storage of a desktop computer. iPhone and mobile device applications are really good for simple, limited-purpose functions, but they should not do everything that a desktop application does.
- The iPhone simulator is a fine tool, but don't limit testing to the iPhone simulator. A simulator is just a simulator. There is a keyboard and a mouse associated with the iPhone simulator. To really test a complicated design, the application must be tested from a physical iPhone.

APPLE IPHONE SDK TOOLS

When the iPhone originally shipped, you could not run third-party native applications directly on the device — until March 6, 2008, when Apple released the first beta of the SDK. The iPhone SDK allows third parties to write applications and run them natively on the device. Since that date, there have been a steady stream of updated beta and released versions of the iPhone SDK. Originally, the iPhone SDK supported both the iPhone and the iPod Touch. With the beta release of the iPhone SDK Version 3.2, Apple added support for the iPad tablet device.

Tools

The Apple SDK contains a number of tools that are important to the MonoTouch developer. These tools are:

- **Xcode:** A suite of tools for development in an Apple environment, the main tool being the IDE. Although MonoTouch does not directly use the Xcode IDE, it can help you create a simple app to deploy to a device. You can also use it to verify that the certificates and provisioning information on the associated devices are working properly.
- **Interface Builder:** Interface Builder (IB) allows for the graphical creation of a user interface. The MonoDevelop IDE integrates with IB and converts the interface created within IB into a user interface callable by MonoTouch.

- **Simulator:** Allows for emulating the iPhone, iPod Touch, and the iPad. Note that the simulator does not run ARM code. It runs x86 code.
- **Libraries necessary to target the device:** This includes libraries for Cocoa Touch, audio, video, networking, SQLite, threads, power management, and the general OS X Kernel.

Licensing

The SDK is a free download. Unfortunately, to release software for the iPhone, a developer must join the iPhone Development Program. At the time of this writing, the cost to join is $99 (U.S. dollars) a year. The cost of joining varies from country to country. The ability to distribute applications to devices is dependent on having the necessary development certificates. These are available through the Apple Developer site once a developer joins the iPhone Developer Program.

SUMMARY

This chapter looked at the following items in the marketplace:

- The iPhone, its licensing, and its operating system
- The .NET Framework and Mono
- MonoTouch, which allows .NET developers to target the iPhone
- MonoDevelop, which allows developers to have a good IDE to write code with MonoTouch

You should now be familiar with which tools are needed to build a native application with .NET/C# for the iPhone. The next chapter explores the specifics of building a MonoTouch application with MonoDevelop. Chapters 3 and 4 describe how to work with the user controls for user input and for presenting data to a user in a standard form factor. Other chapters in the book will explore specific parts of the iPhone, such as maps, acceleration, and the iPad.